RESEARCH PAPER

Making Canonical Workflow Building Blocks Interoperable across Workflow Languages

Stian Soiland-Reyes[1,2†], Genís Bayarri[3], Pau Andrio[4], Robin Long[5,6], Douglas Lowe[6], Ania Niewielska[7], Adam Hospital[3] & Paul Groth[2]

[1]Department of Computer Science, The University of Manchester, Manchester M13 9PL, UK

[2]Informatics Institute, University of Amsterdam, Amsterdam 1098 XH, The Netherlands

[3]Institute for Research in Biomedicine (IRB Barcelona), The Barcelona Institute of Science and Technology (BIST), Barcelona 08028, Spain

[4]The Spanish National Bioinformatics Institute (INB), Barcelona Supercomputing Center (BSC), Barcelona 08034, Spain

[5]Data Science Institute, Lancaster University, Lancaster, Lancashire LA1 4YW, UK

[6]Research IT, IT Services, The University of Manchester, Manchester M13 9PL, UK

[7]European Bioinformatics Institute (EMBL-EBI), Hinxton, Cambridgeshire CB10 1SD, UK

Keywords: Scientific workflows; Interoperable; FAIR; Computational tools; Containers; Software packaging; FAIR digital object (FDO), BioExcel Building Blocks library (BioBB), Canonical Workflow Frameworks for Research (CWFR)

Citation: Soiland-Reyes, S., et al.: Making canonical workflow building blocks interoperable across workflow languages. Data Intelligence 4(2), 342-357 (2022). doi: 10.1162/dint_a_00135

Received: October 1, 2021; Revised: November 26, 2021; Accepted: February 5, 2022

ABSTRACT

We introduce the concept of *Canonical Workflow Building Blocks* (CWBB), a methodology of describing and wrapping computational tools, in order for them to be utilised in a reproducible manner from multiple workflow languages and execution platforms. The concept is implemented and demonstrated with the BioExcel Building Blocks library (BioBB), a collection of tool wrappers in the field of computational biomolecular simulation. Interoperability across different workflow languages is showcased through a protein Molecular Dynamics setup transversal workflow, built using this library and run with 5 different Workflow Manager Systems (WfMS). We argue such practice is a necessary requirement for FAIR Computational Workflows and an element of Canonical Workflow Frameworks for Research (CWFR) in order to improve widespread adoption and reuse of computational methods across workflow language barriers.

† Corresponding author: Stian Soiland-Reyes (Email: soiland-reyes@manchester.ac.uk; ORCID: 0000-0001-9842-9718).

1. INTRODUCTION

The need for *reproducibility* of research software usage is well established [1, 2, 3], and adaptation of *workflow management systems* (WfMS) together with *software packaging and containers* [4] have been proposed as key ingredients for making research software usage *FAIR* and reproducible [5, 6, 7]. Recently it is also argued that computational workflows should also be treated as FAIR Digital Objects [8] in their own right, with identifier, metadata [2] and interoperability requirements [9].

BioExcel, a European Centre of Excellence for Computational Biomolecular Research, has a particular focus on the research domains molecular dynamics simulations and bioinformatics with use of *High Performance Computing* (HPC) to approach Exascale performance, while also improving usability. The *BioExcel Building Blocks* (BioBB) [10] have been created as portable wrappers of open-source computational tools identified as useful for BioExcel workflows, forming several *families* of documented and interoperable operations that can be called from multiple workflow systems. This interoperability is shown with the BioBB demonstrator workflows, along with multiple tutorials and notebooks.

We propose that these building blocks and their families themselves can be considered *composite Digital Objects*: collections of software packages and their source code, guides and tutorials, as well as workflow management system integrations and workflow examples. In addition, the building blocks, as wrappers of upstream open source tools, benefit from and refer to the tools' existing documentation, support forums, academic publications and wider development context.

Given BioBB as a starting point, we define a generalized methodology of *Canonical Workflow Building Blocks* (**CWBB**), through the definition of a set of requirements and recommendations for how to formalize and develop a family of compatible computational tools as Digital Objects. These building blocks let researchers instantiate a Canonical Workflow in multiple workflow management systems, while also benefiting from the FAIR aspects of the CWBB Digital Objects.

2. METHODS

The BioExcel Building Blocks library [10], created and implemented within the BioExcel CoE, is a collection of portable wrappers of common biomolecular simulation tools. The BioBB library is designed to i) increase the *interoperability* between the tools wrapped; ii) *ease the implementation* of biomolecular simulation workflows; and iii) increase the *reusability and reproducibility* of the generated workflows. To achieve these main goals, the library was designed following the FAIR principles for research software development best practices [7].

The result is a collection of building block modules, divided in sets of tool wrappers focused on similar functionalities (e.g. Molecular Dynamics, Virtual Screening). Each of the modules is built from a combination of (i) software packaging (Pip, BioConda, BioContainers), (ii) documentation (ReadTheDocs), (iii) interactive tutorials (Jupyter Notebooks, Binder), (iv) registry & findability (bio.tools, BioSchemas, WorkflowHub), (v)

WfMS integration stubs (CWL, Galaxy, PyCOMPSs), (vi) source Code (GitHub) and (vii) REST APIs (OpenAPI, Swagger). Notably all building blocks follow the same pattern of installation, configuration and interaction.

Since the publication of the library, several new building block modules (Chemistry, Machine Learning, AMBER MD, Virtual Screening, etc.) have been added, and the set of operations for the existing BioBB families have been expanded. While we previously provided curated adapters (Figure 1) for running BioBB in workflow systems using Common Workflow Language (CWL) and PyCOMPSs, along with Galaxy Toolshed bindings, we have now started auto-generating these bindings, along with command line wrappers and REST web service APIs, using annotations within BioBB's Python docstrings as source. These annotations include sufficient information for a WfMS to launch a particular building block: *input* and *output* parameters (including *mandatory/optional* flags), compatible *formats* (including from EDAM ontology [11]), *example* files (essential for testing purposes), default values and *dependencies*. This ensures human-readable documentation, FAIR metadata and programmatic accessibility can be generated consistently and comparably.

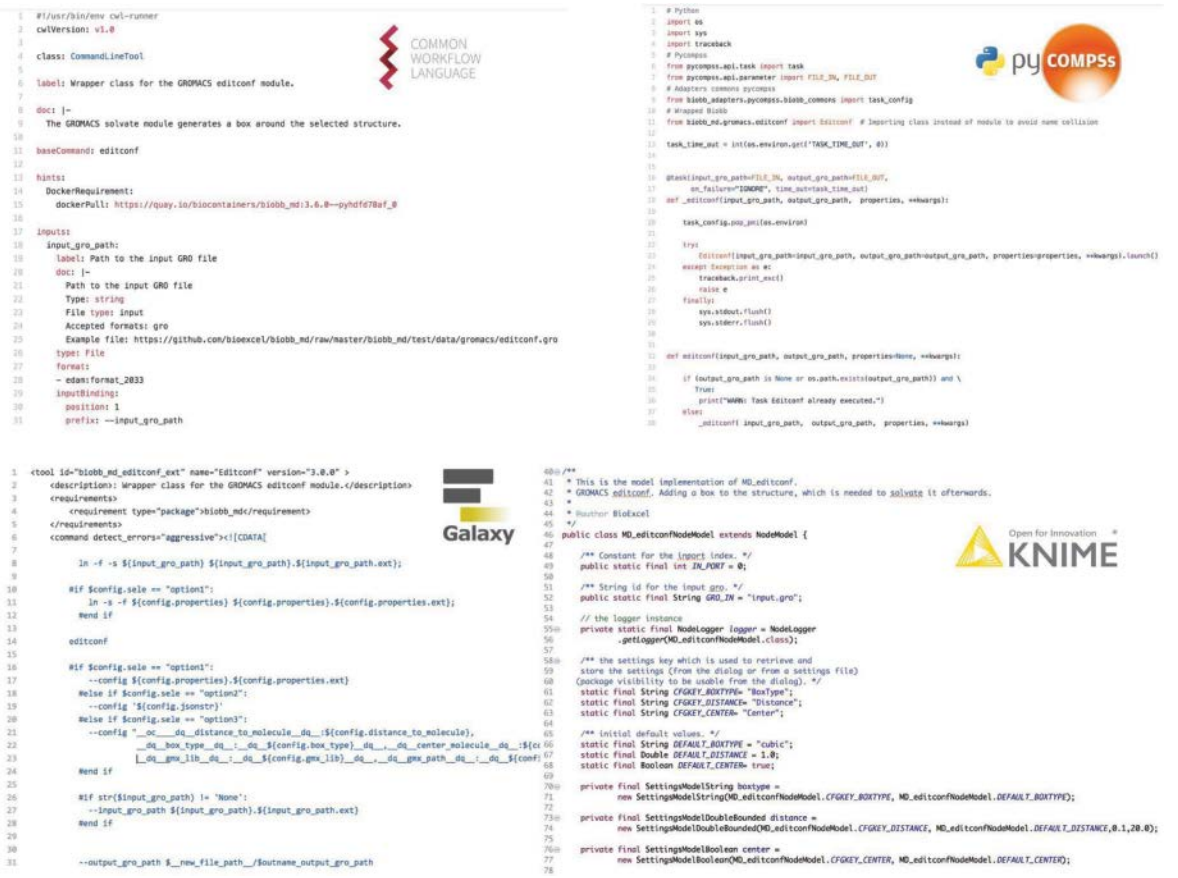

Figure 1. Code snippets for the BioBB WfMS bindings: CWL, PyCOMPSs, Galaxy and KNIME.

The library is showcased through a collection of demonstration workflows [12]. Here, each workflow introduces individual building blocks as needed to explain a particular scientific computational method. We primarily expose the workflows as Jupyter Notebooks [13], which has been highlighted as a valuable tool for reproducible scientific workflows [14]. This offers a graphical interactive interface, including documentation (integrated markdown) related to the workflow and the building blocks used, but also to the biomolecular simulation methods used in the pipeline. Moreover, as we have demonstrated with our own Binder [15] hosting, these workflows are reproducible across platforms, assisted by BioConda [16] packaging of the building blocks and their software dependencies.

This assembly of available demonstration workflows has been successfully used in the BioExcel CoE for dissemination with a range of training events (e.g., BioExcel Summer & Winter School, webinars and virtual training). In training we particularly utilised the Binder infrastructure of the BioExcel Cloud portal [17] to give users a web-based first experience of the building blocks before they try them in other workflow systems.

We can observe that workflow building blocks such as BioBB are necessarily composed of a comprehensive list of digital objects, encompassing source code, packaging, containerization, documentation, attributions, citations, registry entries, WfMS integrations and REST APIs.

We propose to consider building blocks as *composite digital objects* in their own right: gathering the above software components along with their metadata, identifiers and operations then forms a *Canonical Workflow Building Block* (CWBB). We suggest this concept as a fundamental element of FAIR Digital Objects for Computational Workflows: researchers use the building blocks computationally as functional operations across WfMSs, while the FAIR aspect of CWBB propagates information and resources that are essential for reproducibility, reuse and understanding by anyone discovering the workflow.

2.1 Interoperability across Different Workflow Languages

The concept of Canonical Workflow Building Blocks is here showcased with the BioBB library, by using a transversal workflow present in many different computational biomolecular projects: a Molecular Dynamics (MD) protein setup. This workflow prepares a protein structure to be used as input for an MD simulation, going through a series of steps where the protein is completed (adding hydrogen and missing atoms), optionally introducing a residue mutation, then submerging the protein in a virtual box of water molecules with a particular ionic concentration, and finally energetically equilibrating the system (so that solvent and ions are well accommodated around the protein at the desired temperature).

This simulation process involves a non-negligible number of steps, using a variety of biomolecular tools. The BioBB library was used to assemble this workflow, interconnecting building blocks using Python functions (Jupyter Notebook, Command Line Interface), auto-generated bindings (Galaxy [18], CWL [19], PyCOMPSs [20]) or manually generated bindings (KNIME [21]). Corresponding workflows for the different WfMS can be found in WorkflowHub [22, 23, 24, 25, 26] and graphical extracts can be seen in Figure 2.

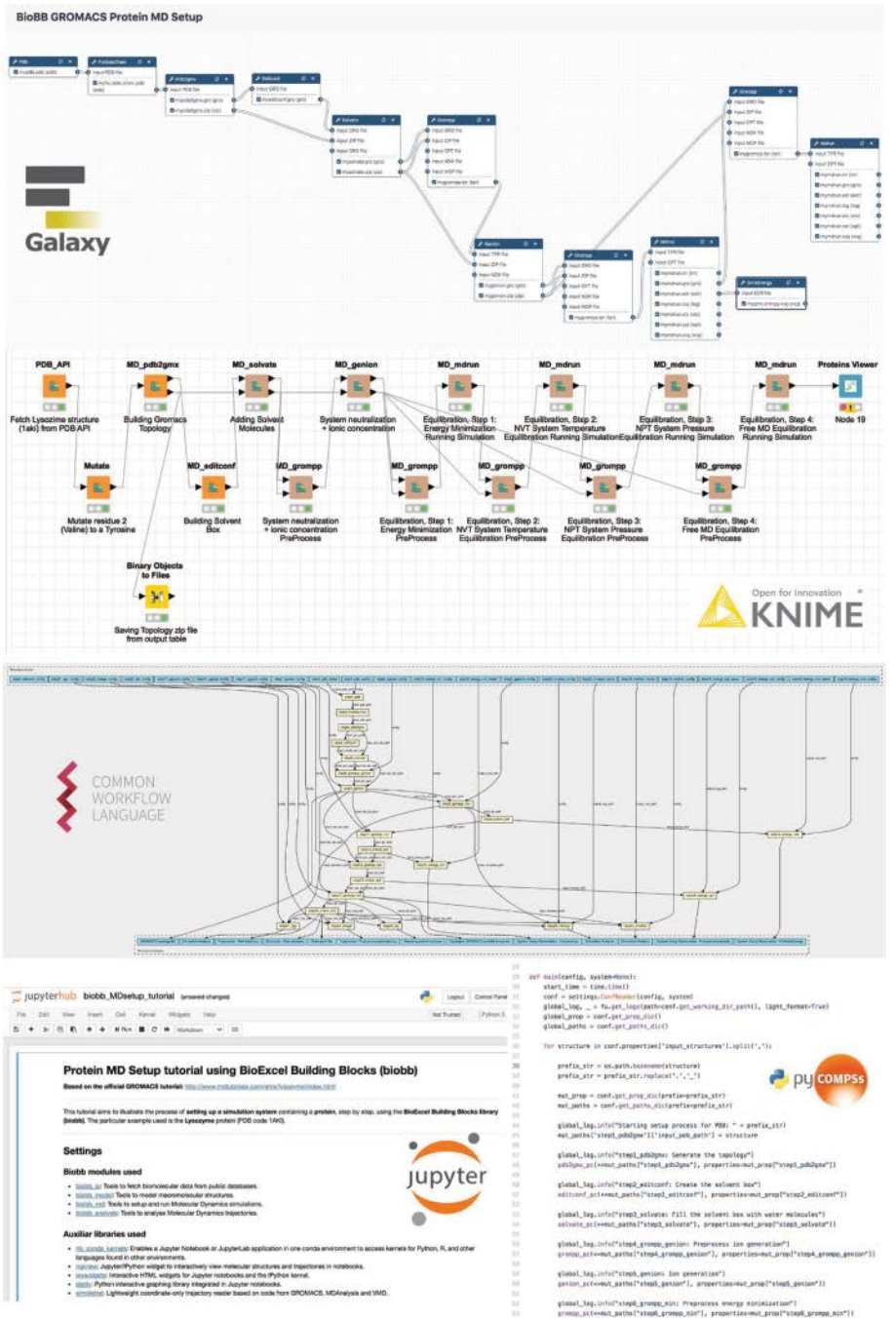

Figure 2. Protein MD Setup transversal workflow, assembled in with 5 different workflow managers using BioBB canonical building blocks. From top-left: Galaxy [22], KNIME [23], CWL [24], Jupyter Notebook [25], and PyCOMPSs [26].

This example demonstrates how the same canonical building blocks can be used in different WfMS. Wrappers and tools executed behind the workflows are exactly the same, but the workflows are built using different WfMS, some of them in a graphical way (Galaxy, KNIME), some in a command line way (Jupyter Notebook, PyCOMPSs, CWL); workflows can be focused on short/interactive executions (Jupyter Notebook), or on High Throughput/High Performance Computing (HT-HPC) executions (PyCOMPSs); some of them prepared for a particular WfMS installation (Galaxy), others completely system-agnostic (CWL).

The current WfMS bindings include Jupyter Notebook, PyCOMPSs, CWL, Galaxy and KNIME WfMS, in addition to a command line mechanism. Thanks to the extensive documentation added in the source code as Python docstrings, new bindings for available WfMS can be generated. We are also experimenting with generating a REST API exposing the building services as Web services. However, it should be noted that such automatic generation of bindings is not always practically feasible. As an example, KNIME nodes require a complete Java skeleton code, as well as a definition of new data types for all inputs/outputs required, which makes their automatic generation a heavy and potentially error-prone task. Bindings for workflow languages with a *domain-specific language* (DSL) for tool definitions (e.g. Galaxy, CWL) can, on the other hand, be generated in a more straightforward fashion.

The transversal protein MD setup workflow was chosen as a real example that is readily understandable by domain experts. More complex pipelines involving a broader set of wrapped biomolecular tools have been developed using the BioBB library, primarily as Jupyter Notebooks. A selection of these have similarly been assembled for different WfMS using the auto-generated bindings and uploaded to the WorkflowHub repository.

3. DISCUSSION

Early work on libraries of workflows fragments include Web Service-based approaches where tools are wrapped and exposed using common, interoperable data types in BioMoby [27] for bioinformatics and similarly caBIG [28] for cancer genomics. While these efforts were interoperable across WfMSs they required a large up-front investment in agreeing to and adapting native data to common RDF or XML representations.

The notion of *abstract workflows* [29], structural workflow descriptions separated from their concrete execution realisations and augmented with Linked Data annotations, have been emphasised as essential for reuse and consistency across workflow systems. Identifying *common motifs* for workflow operations [30] (e.g. Data preparation, Format transformation, Filter, Combine) are important to simplify and understand otherwise fine-grained workflow provenance traces.

Most other efforts to standardise a set of disparate analytical tools have been done within the scope of a single WfMS, allowing customised user interaction, data visualisation, configuration and findability, for instance Taverna components had prototypical building blocks [31] which were instantiated at runtime by reference from a registry. KNIME components and metanodes, shared on the KNIME Hub are frequently designed

to be interoperable, but with a perhaps weaker notion of component families. The Galaxy toolshed [32] is likewise populated with different sets of tool wrappers that are largely made to be interoperable within a category.

The *Common Workflow Language* (CWL) [19] has a strong emphasis on interoperable command line tool descriptions, with support for containers and Conda packaging, as well as support for FAIR metadata like contributors, license and EDAM ontology type annotations. With multiple leading workflow engines now supporting CWL, and experimental Galaxy support, this seems perhaps the most promising candidate for both making and describing canonical workflow building blocks, however we've identified a few stumbling blocks.

One obvious challenge is that the implementing WfMS needs to have CWL support, along with support for either containers or Conda packaging to find the described executables. While it is possible to run a CWL tool directly using a #!/usr/bin/env cwl-runner shebang on POSIX systems, this still requires pre-installation and possibly configuration of a CWL engine like cwltool or Toil [33]. However workflow engines have multiple dependencies and often cannot easily be run from a container themselves[①].

Within the CWL community it was originally envisioned that a wider set of workflow systems would adopt CWL for tool description/execution, with a subset implementing full CWL workflow support. This would allow shared community effort for describing tools, say in the Common Workflow Library, rather than each WfMS needing to duplicate this tool wrapping in separate repositories and languages. However, with the exception of experimental tool support in Galaxy, in practice all CWL implementers have gone for full workflow support.

Another challenge is that making a set of building blocks frequently requires the use of *shims*, for instance file conversion, small search/replace operations or file renames. In a CWL approach these can either be performed with an Expression using JavaScript snippets which only has limited access to file content, or as an additional workflow step added before or after the main tool step. This combination could then be nested as a subworkflow, similar to KNIME's *metanodes*, and would also be flexible by allowing different containers or packages for any pre- or post-steps. Such a CWL building block however becomes harder to access from a non-CWL WfMS, because of lack of control over configuration/execution options for the now nested CWL tools. In practice[②], executing a nested CWL workflow from a native WfMS language would require the engine to implement full CWL Workflow support (or delegate to a CWL engine).

For the main BioBB building blocks we implemented demonstrator workflows that highlight how the tools should be used in different workflow management systems; each having a primary exemplar using Jupyter Notebook, which can be explored interactively using the BioExcel Binder. If we consider the abstract

[①] To execute the wrapped tool, a containerized workflow engine would need *nested containers* which are not generally recommended for security reasons. It is possible to work around this limitation using Singularity or Conda.

[②] It is worth mentioning that it would also be possible to generate WfMS-specific bindings from CWL descriptions (e.g. as demonstrated with cwl2script for Bash, gxargparse for Galaxy, cwl2wdl for WDL), although this necessitates constraining the tool and workflow definitions to a limited mappable subset of CWL.

demonstrator workflows as *canonical workflows* they are therefore very much active objects, but can also be seen as *workflow templates*, as any real use case will need to specialise the workflow to tweak parameters, data selection etc.

We therefore also provide such workflow templates for multiple WfMS, including CWL, PyCOMPSs and Galaxy. These are fairly disparate workflow languages, yet by the use of the same canonical workflow building blocks (which again invoke the same software binaries), such WfMS-specific workflows effectively are instantiations of the same canonical workflow.

One challenge found is how to publish such canonical workflows in registries like the WorkflowHub. The hub supports the registration of Digital Objects in the form of RO-Crate [34], with the option of abstract CWL for describing the canonical workflow template, along with direct references to the workflow's GitHub repository.

For instance in the RO-Crate for https://doi.org/10.48546/workflowhub.workflow.200.1 [26], which can also be rendered from GitHub, we have an entry for the *main workflow* according to the Workflow RO-Crate profile, detailing each canonical workflow building block used (e.g., biobb-md metadata). Here the FAIR aspect of the building blocks to help software citation is exercised, as the building block wrapper has one set of authors, documentation and licence (Apache-2.0), while the wrapped software (e.g. GROMACS metadata) has different authors, licence (GPL-2.1+) and documentation.

However, the deposit of such RO-Crates in WorkflowHub results in one registration entry per workflow language, which are not otherwise related and may not even share the same source code repository. Thus, we've identified the need for adding an overall *canonical workflow entry*, which can bring in workflow documentation and references shared across WfMS implementations, including a set of links to the more granular canonical workflow building blocks used by the workflow, but also to the individual WfMS implementations as separate digital objects.

A similar question of granularity applies at the workflow tool level [4], particularly for Findability and Accessibility, as we can consider at lowest granularity the *scientific method* in general (e.g., any algorithm for sequence alignment), followed by an *application suite* (bio.tools entry [35], homepage, documentation), instantiated as a particular *software installation* (Debian package, Docker container) with its dependencies at same level. The installation includes one or more *software executables* (a particular binary, a running service), providing at the highest detailed granularity level the specific types of *software functionality* (a particular mode of operation, choice of analysis), for instance using certain command line flags.

For canonical workflow building blocks, with a focus on pluggable composability, this is mainly defined at this high granularity level of specific software functionality: explicit operations from an installed tool, which are then combined in a workflow. This is indeed the level WfMS tool definitions are typically done, e.g., a CWL Command Line Tool specifies a particular way to run a particular software binary. However, to be an actionable CWBB, the building block needs to additionally convey the lower granularity levels; particularly to support multiple options for interoperable installation and execution, as well as metadata at the most general level, such as documentation and scholarly citations.

While workflow management systems typically only operate at the highest granularity levels for execution details, and are frequently unaware of (or not exposing metadata at) the more general levels, we argue that in order for a Canonical Workflow [36] to follow and support FAIR principles for itself and its data, the workflow management system needs to *propagate structured metadata* about the tools used by the workflow. We propose that in order to support the workflow's applicability to multiple WfMS, the tools themselves must also have a consistent packaging and formal description that enables computational invocation.

At the most general level, a canonical workflow built using such CWBBs is even conceptually reproducible because the FAIR documentation of the workflow, through its canonical workflow building blocks, identifies how individual tools and software applications are composed, which in worst case can be rebuilt using different installation methods in a different WfMS, or in best case inspected to detect and cross-link the same canonical workflow appearing in different WfMS instantiations. This view of software as composition of other software typically also applies at individual tool level, which themselves depend on programming language runtimes, libraries, services and reference data.

4. REQUIREMENTS FOR CANONICAL WORKFLOW BUILDING BLOCKS

Building on the experiences with BioBB, we here propose requirements and recommendations for establishing Canonical Workflow Building Blocks (CWBB) as implementations of *canonical steps* introduced for Canonical Workflow Frameworks for Research [36].

The core purpose of a CWBB is to wrap a command line tool or other software that can perform an operation as part of a computational workflow. As such, the general advice for making software workflow-ready applies [37] (e.g., easy to install, documented, parallelizable, reproducible output), however a CWBB is also permitted to make use of additional scripts or *shims* to further adapt a third-party tool for workflow use and for data interoperability across blocks.

The way tools are installed or invoked varies slightly across WfMS and operating systems, therefore a CWBB should provide multiple methods for distributing software; currently containers (Docker, Singularity) and distribution-independent packaging (e.g., Conda, Homebrew) are promising by having reproducible install recipes and a wide range of available open source dependencies (e.g. Java, Python). Additionally building blocks should allow overriding execution paths, e.g., for use with HPC module system and hardware-optimised binaries.

The CWBBs should have sufficient annotations to be able to generate bindings for different WfMSs and REST APIs, e.g. parameter names and descriptions, types and default values; enumerators for options, file formats for inputs/outputs.

Building blocks should be grouped into families that are interoperable through common data structures and file formats, as well as having joint naming conventions for configuration options. A CWBB family should be released as a single version following semantic versioning rules, which should have a corresponding persistent identifier (PID) [38].

Metadata for CWBBs should be captured following FAIR guidelines, and distributed as part of the block family and resolvable from the PID as a FAIR Digital Object. Metadata should include references to the CWBB software distributions (e.g., quay.io container URL) as well as attributions, citations and documentation for the wrapped tool.

Example workflows showing CWBB usage should be included in a WfMS-neutral language such as Jupyter Notebooks, which may have equivalent variants for each workflow binding. These workflows should be registered in a workflow registry like WorkflowHub or Dockstore, and assigned their own PIDs.

5. CONCLUSIONS

The proposed concept of Canonical Workflow Building Blocks can bridge the gap between FAIR Computational Workflows, interoperable reproducibility and for building canonical workflow descriptions to be used and described FAIRly across WfMSs.

The realisation of CWBBs can be achieved in many ways, not necessarily using the Python programming language together with RO-Crate as explored here. In particular, if the envisioned Canonical Workflow Frameworks for Research become established in multiple WfMSs with the use of FAIR Digital Objects, the different implementations will need to agree on object types, software packaging and metadata formats in order to reuse tools and provide interoperable reproducibility for canonical workflows.

Likewise, to build a meaningful collection of building blocks for a given research domain, a directed collaborative effort is needed to consistently wrap tools for a related set of WfMSs, chosen to target particular use cases (a family of canonical workflows).

For individual users, a library of Canonical Workflow Building Blocks simplifies many aspects of building pipelines, beyond the FAIR aspects and data compatibility across blocks. For instance, they can benefit from training of a CWBB family using Jupyter Notebooks, and then use this knowledge to utilise the same building blocks in a scalable HPC workflow with a CWL engine like Toil, knowing they will perform consistently thanks to the use of containers.

While we have demonstrated CWBB in the biomedical domain, this approach is generally applicable to a wide range of sciences that execute pipelines of multiple file-based command line tools, however it may be harder to achieve with more algebraic “in memory” types of computational workflows, where steps could be challenging to containerize and distinguish as separate block.

We admit that biomolecular research is quite a homogenous field with respect to computational analyses and now becoming relatively mature in terms of tool composability in workflows, building on the experiences of the “FAIR pioneers” in the field of bioinformatics. Other fields, such as social sciences or ecology, can have a wider variety of methods and computational tools, often with human interactions, and may have to adapt the software to be workflow-ready [37] before using them as Canonical Workflow Building Blocks. Domains adapting CWBB approach (or workflow systems in general) should take note of the great benefits

of hosting collaborative events where developers meet each other and their potential users, demonstrated in our field with events such WorkflowsRI [39] and Biohackathons [40].

The Common Workflow Language shows promise as a general canonical workflow building blocks mechanism: gathering execution details of tools along with their metadata and references, augmented with abstract workflows to represent canonical workflows. However, this would need further work to implement our CWBB recommendations in full. Future work for the Canonical Workflow Building Blocks concept includes formalising and automating publication practises, to make individual blocks available as FAIR Digital Objects on their own or as part of an aggregate collection like RO-Crate.

ACKNOWLEDGEMENTS

This work has been done as part of the BioExcel CoE (https://www.bioexcel.eu/), a project funded by the European Union contracts H2020-INFRAEDI-02-2018 823830, and H2020-EINFRA-2015-1 675728. Additional work is funded through EOSC-Life (https://www.eosc-life.eu/) contract H2020-INFRAEOSC-2018-2 824087, and ELIXIR-CONVERGE (https://elixir-europe.org/) contract H2020-INFRADEV-2019-2 871075.

AUTHOR CONTRIBUTIONS

Stian Soiland-Reyes (soiland-reyes@manchester.ac.uk): Conceptualization, Funding acquisition, Investigation, Methodology, Project administration, Supervision, Writing—original draft, Writing—review & editing

Genís Bayarri (genis.bayarri@irbbarcelona.org): Software, Software Documentation

Pau Andrio (andriopau@gmail.com): Methodology, Software, Validation, Software Documentation

Robin Long (r.long@lancaster.ac.uk): Software, Software Documentation

Douglas Lowe (douglas.lowe@manchester.ac.uk): Software, Software Documentation

Ania Niewielska (aniewielska@ebi.ac.uk): Methodology, Resources, Software

Adam Hospital (adam.hospital@irbbarcelona.org): Methodology, Project administration, Resources, Software, Validation, Visualization, Writing—original draft, Writing—review & editing

Paul Groth (p.t.groth@uva.nl): Supervision, Writing—review & editing

The authors would also like to acknowledge contributions from: Felix Amaladoss, Cibin Sadasivan Baby, Finn Bacall, Rosa M. Badia, Sarah Butcher, Gerard Capes, Michael R. Crusoe, Alberto Eusebi, Carole Goble, Josep Lluís Gelpí, Modesto Orozco, and Geoff Williams.

REFERENCES

[1] Stodden, V., et al.: Enhancing reproducibility for computational methods. Science 354(6317), 1240–1241 (2016)

[2] Leipzig, J., et al.: The role of metadata in reproducible computational research. Patterns 2(9), 100322 (2021)

[3] Katz, D.S., et al.: A fresh look at FAIR for research software. arXiv preprint arXiv:2101.10883 (2021)

[4] Möller, S., et al.: Robust cross-platform workflows: How technical and scientific communities collaborate to develop, test and share best practices for data analysis. Data Science and Engineering 2, 232–244 (2017)

[5] Cohen-Boulakia, S., et al.: Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities. Future Generation Computer Systems 75, 284–298 (2017)

[6] Grüning, B., et al.: Practical computational reproducibility in the life sciences. Cell Systems 6(6), 631–635 (2018)

[7] Lamprecht, A.-L., et al.: Towards FAIR principles for research software. Data Science 3(1), 37–59 (2020)

[8] De Smedt, K., Koureas, D., Wittenburg, P.: FAIR digital objects for science: From data pieces to actionable knowledge units. Publications 8(2), 21 (2020)

[9] Goble, C., et al.: (2020): FAIR Computational Workflows. Data Intelligence 2(1), 108–121 (2020)

[10] Andrio, P., et al.: BioExcel building blocks, a software library for interoperable biomolecular simulation workflows. Scientific Data 6, 169 (2019)

[11] Ison, J., et al.: EDAM: An ontology of bioinformatics operations, types of data and identifiers, topics and formats. Bioinformatics 29(10), 1325–1332 (2013)

[12] Hospital, A., et al.: BioExcel-2 Deliverable 2.3—First release of demonstration workflows (2020). Available at: https://doi.org/10.5281/zenodo.4540432. Accessed 26 November 2021

[13] Kluyver, T., et al.: (2016): Jupyter notebooks—a publishing format for reproducible computational workflows. In: Loizides, F., Schmidt, B. (eds.) Positioning and Power in Academic Publishing: Players, Agents and Agendas, pp 87–90. Available at: https://doi.org/10.3233/978-1-61499-649-1-87. Accessed 26 November 2021

[14] Beg, M., et al.: Using Jupyter for reproducible scientific workflows. Computing in Science & Engineering 23(2), 36–46 (2021)

[15] Jupyter Project, Bussonnier, M., et al: Binder 2.0—Reproducible, interactive, sharable environments for science at scale. In: Proceedings of the 17th Python in Science Conference (SciPy 2018), pp. 113–120 (2018)

[16] Grüning, B., et al.: Bioconda: Sustainable and comprehensive software distribution for the life sciences. Nature Methods 15, 475–476 (2018)

[17] Niewielska, A., Butcher, S., Westermaier, Y.: BioExcel-2 Deliverable 2.5—Provision of a workflow environment at BioExcel portal. Available at: https://doi.org/10.5281/zenodo.4916060. Accessed 26 November 2021

[18] Afgan, E., et al.: The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. Nucleic Acids Research 46(W1), W537–W544 (2018)

[19] Crusoe, M.R., et al.: Methods included: Standardizing computational reuse and portability with the common workflow language. Communication of the ACM (in press). Available at: https://doi.org/10.1145/3486897. Accessed 26 November 2021

[20] Tejedor, E., et al.: PyCOMPSs: Parallel computational workflows in Python. The International Journal of High Performance Computing Applications 31(1), 66–82 (2017)

[21] Fillbrunn, A., et al.: KNIME for reproducible cross-domain analysis of life science data. Journal of Biotechnology 261, 149–156 (2017)

[22] Lowe, D.: Protein MD setup tutorial using BioExcel building blocks (biobb) in Galaxy. WorkflowHub. Workflow (Galaxy). Available at: https://doi.org/10.48546/workflowhub.workflow.194.1. Accessed 26 November 2021

[23] Hospital, A.: Protein MD setup tutorial using BioExcel building blocks (biobb) in KNIME. WorkflowHub. Workflow (KNIME). Available at: https://doi.org/10.48546/workflowhub.workflow.201.1. Accessed 26 November 2021
[24] Bayarri, G., Long, R.: Protein MD setup tutorial using BioExcel building blocks (biobb) in CWL. WorkflowHub. Workflow (CWL). Available at: https://doi.org/10.48546/workflowhub.workflow.29.3. Accessed 26 November 2021
[25] Bayarri, G., Hospital, A., Lowe, D.: (2021): Protein MD setup tutorial using BioExcel building blocks (biobb) in Jupyter Notebook. WorkflowHub. Workflow (Jupyter Notebook). Available at: https://doi.org/10.48546/workflowhub.workflow.120.2. Accessed 26 November 2021
[26] Hospital, A., Andrio, P.: Protein MD setup HPC tutorial using BioExcel building blocks (biobb) in PyCOMPSs. WorkflowHub. Workflow (PyCOMPSs). Available at: https://doi.org/10.48546/workflowhub.workflow.200.1. Accessed 26 November 2021
[27] The BioMoby Consortium: Interoperability with Moby 1.0—It's better than sharing your toothbrush! Briefings in Bioinformatics 9(3), 220–231 (2008)
[28] Saltz, J., et al.: caGrid: Design and implementation of the core architecture of the cancer biomedical informatics grid. Bioinformatics 22(15), 1910–1916 (2006)
[29] Garijo, D., Gil, Y.: A new approach for publishing workflows. In: Proceedings of the 6th Workshop on Workflows in Support of Large-Scale Science—WORKS '11, pp. 47–56 (2011)
[30] Garijo, D., et al.: Common motifs in scientific workflows: An empirical analysis. Future generation computer systems 36, 338–351 (2014)
[31] De Giovanni, R., et al.: ENM components: A new set of Web service - based workflow components for ecological niche modelling. Ecography 39(4), 376–383 (2016)
[32] Blankenberg, D., et al.: Dissemination of scientific software with Galaxy ToolShed. Genome Biology 15: Article No. 403 (2014)
[33] Vivian, J., et al.: Toil enables reproducible, open source, big biomedical data analyses. Nature Biotechnology 35 pp 314–316 (2017)
[34] Soiland-Reyes, S., et al.: Packaging research artefacts with RO-Crate. Data Science (in press). Available at: https://doi.org/10.3233/DS-210053. Accessed 26 November 2021
[35] Ison, J., et al.: biotoolsSchema: A formalized schema for bioinformatics software description. GigaScience, 10(1): giaa157 (2021)
[36] The CWFR Group, Hardisty, A., (ed.), Wittenburg, P., (ed.): CWFR position paper. OSF, January 6, 2021. Available at: https://osf.io/3rekv/. Accessed 26 November 2021
[37] Brack, P., et al.: 10 simple rules for making a software tool workflow-ready. PLOS Computational Biology 18(3), e1009823 (in press). Available at: https://doi.org/10.1371/journal.pcbi.1009823. Accessed 26 November 2021
[38] McMurry, J.A., et al.: Identifiers for the 21st century: How to design, provision, and reuse identifiers to maximize utility and impact of life science data. PLOS Biology 15(6): e2001414 (2017)
[39] Ferreira da Silva, R., et al.: A community roadmap for scientific workflows research and development. arXiv preprint arXiv: 2110.02168 (2021)
[40] Garcia, L., et al.: Ten simple rules to run a successful BioHackathon. PLOS Computational Biology 16(5): e1007808 (2020)

AUTHOR BIOGRAPHY

Stian Soiland-Reyes is a Technical Architect in the eScience Lab at The University of Manchester, and a Ph.D. Candidate in the INDE Lab at University of Amsterdam. Since 2006 he has worked as a software engineer and researcher focusing on reproducibility, scientific workflows, interoperability, linked data, metadata and open science. He is a persistent advocate of Open Scholarly Communication, and is on the leadership teams of Common Workflow Language and BioCompute Objects. He contributed to the W3C provenance model PROV-O and multiple Linked Data initiatives. He co-created the Research Object model and is co-chair of the Research Object Crate community.
ORCID: 0000-0001-9842-9718

Genís Bayarri is a Full Stack Developer working as a Web Developer and Software Research Engineer in the Molecular Modeling and Bioinformatics group of the Institute for Research in Biomedicine in Barcelona (IRB Barcelona). He graduated in Multimedia Engineering, and has a broad experience in the private sector. He joined the IRB Barcelona in 2016 and since then he has participated in the European projects BioExcel Centre of Excellence and Multiscale Complex Genomics. He has also developed a set of public Web servers, databases and applications related to molecular dynamics, specially in the 3D representation field.
ORCID: 0000-0003-0513-0288

Pau Andrio is a Research Software Engineer (RSA) working for the Spanish National Bioinformatics Institute (INB) at the Barcelona Supercomputing Center (BSC). He graduated from the Universitat Jaume I (UJI) with a Bachelor's degree in Computer Sciences and a Master's degree in Biomedicine from Universitat de Barcelona (UB) in 2012. Since 2010, he has been involved in several projects in the field of bioinformatics, structural biology and molecular dynamics simulation, also contributing to two main EU projects: Scalalife and BioExcel. His main interest is to provide the life sciences research community; user-friendly, reproducible and scalable software. One of the results of BioExcel, and the main project he is currently contributing to, is the BioBB (BioExcel Building Blocs) library.
ORCID: 0000-0003-2116-3880

Robin Long is a Research Software Engineer at the Data Science Institute of Lancaster University, where he is providing training in research computing and software skills whilst supporting various research software projects. Previously he worked as a Research Software Engineer at The University of Manchester where he developed and improved software tools used for computational biomolecular research and analysis workflows in BioExcel. He is a 2017 Fellow of the Software Sustainability Institute.
ORCID: 0000-0003-2249-645X

Douglas Lowe is a Research Software Engineer working in the Research-IT department, and eScience Lab at the University of Manchester. He gained his Ph.D. degree in Environmental Sciences at Lancaster University in 2004, and has subsequently worked as a Research Associate at the University of Manchester, focusing on studying air quality around the world using large-scale, open source, atmospheric models. He moved to the Research-IT department in 2019, and has been working with the eScience Lab since 2020, where he has been working on developing scientific workflows in Common Workflow Language using the BioExcel Building Blocks library.
ORCID: 0000-0002-1248-3594

Ania Niewielska is a Lead Software Engineer in the Software Development & Operations team, part of the Technical Services Cluster at EMBL-EBI. She holds a Master's degree in Computer Science from the Silesian University of Technology in Poland. Joining EMBL-EBI in 2017, she brought with her a passion for human-centred software design, as well as extensive experience of system integration and API design from the private sector. More recently she has developed an interest in cloud computing and container orchestration, and their application in scientific workflows. Ania co-leads the development of GA4GH Task Execution Service API. Her team in EMBL-EBI develops and maintains the BioExcel Cloud Portal and its BinderHub component.
ORCID: 0000-0003-0989-3389

Adam Hospital is a Research Software Engineer working as a Research Associate in the Molecular Modeling and Bioinformatics group of the Institute for Research in Biomedicine in Barcelona, Spain. Graduated in Computational Science, he received his PhD in Biotechnology for the University of Barcelona in 2013. Since then, he has been working in computational biomolecular simulation workflows in the Spanish National Institute of Bioinformatics (INB, ELIXIR-ES), and is leading the workflows team of the BioExcel Center of Excellence. Interested in making scientific software, workflows and tools publicly available to the scientific community, he has coordinated and participated in the development of a set of public web servers and databases related to macromolecular structure flexibility.
ORCID: 0000-0002-8291-8071

Paul Groth is Professor of Algorithmic Data Science at the University of Amsterdam where he leads the Intelligent Data Engineering Lab (INDElab) and is scientific director of the university's Data Science Centre. He holds a Ph.D. in Computer Science from the University of Southampton (2007) and has done research at the University of Southern California, the Vrije Universiteit Amsterdam and Elsevier Labs. His research focuses on intelligent systems for dealing with large amounts of diverse contextualized knowledge with a particular focus on Web and science applications. This includes research in data provenance, data integration and knowledge sharing.
ORCID: 0000-0003-0183-6910